Grenoble INP – ENSIMAG
École Nationale Supérieure d' Informatique et de Mathématiques Appliquées

Final master project report

The Australian University - RSISE
Research School of Information Sciences and Engineering

Computation of the Spatial Impulse Response for Ultrasonic Fields on the Graphics Processing Units (GPU)

Luna Florian
3rd year – Option IRV

21st February 2010 – 21st August 2010

RSISE-ANU
North Road - Acton
RSISE Building 115
2601 Canberra - ACT Australia

ANU supervisor
Shams Ramtin, Richard Hartley
FIB supervisor
Pere Brunet

Contacts

Student

Florian Luna
Phone : (+33) 632211964
Adress : Quartier le Plan - Le Grand Barsan - 84110 Vaison-la-Romaine
Email : Florian.Luna@gmail.com

Lab Supervisors

Ramtin Shams
Email : Ramtin.Shams@anu.edu.au
Phone : +61 2 6125 8612
Fax : +61 2 6125 8660

Richard Hartley
Email : Richard.Hartley@anu.edu.au
Phone : (+61) 2 6125 8668
Fax : (+61) 2 6125 8660

Address of the RSISE
ANU College of Engineering & Computer Science
RSISE Building 115, North Road
The Australian National University
Canberra ACT 0200, Australia

FIB tutor

Pere Brunet
pere@lsi.upc.edu

Aknowledgements

First of all, I would like to thank Ramtin Shams for all his support during my internship. He helped me not only for my project but also for settling down in Canberra, which was not an easy task. I would like also to thank Parastoo Sadeghi for her help during my internship. I am very thankful for Richard Hartley who supervised me and contributed to bring me in Canberra.

I am also very thankful for the support of my ENSIMAG tutor Marie-Paule Cani, who supervised me all this year. And my final thank will be to the International Relations Office, Aurélie Bonachera and Marianne Genton who made an outstanding work.

Contents

1 Summary

2 Internship presentation

2.1 Introduction

I chose studying computer science at ENSIMAG because I have always been fascinated by simulating and vizualising our surrounding world through computer devices. That interest led me to choose the Image Processing and Virtual Reality speciality at the Ensimag and then the Interface and Vizualisation speciality at the Polytechnic University of Catalunya. The applications of this topic are growing up and medical imaging is one of the most important. Improving the medical technology by using computer knowledge has always appealed to me. Furthermore, physics of sound is something that I am very interested in due to my musical background. Choosing that internship was not difficult as it was a mix of what I had studied before and personal interests.

The report falls into the following parts. The first one is a presentation of the internship. It is then followed by an analysis of the problem we want to simulate and a brief presentation of CUDA (Computer Unified Architecture). The solution and the development of the application are then detailed. The validity and the results are presented. The report comes to an end with a conclusion about the project achievement and the personal experience gained during the trainee.

2.2 The Research School of Information Science and Engineering

I did my internship at the Research School of Information Sciences and Engineering (RSISE). The RSISE is one of twelve research schools at ANU and was established on 1 January 1994. It evolved from the Department of Systems Engineering (1981) within the then Research School of Physical Sciences and Engineering (RSPhysSE) and the Computer Sciences Laboratory (1988) also within RSPhysSE.

2.3 My supervisors

My two supervisors were Dr Ramtin Shams and Pr Richard Hartley.

Dr. Shams is a Fellow (senior lecturer) at the School of Engineering at the Australian National University (ANU). He received his B.E. and M.E. degrees in electrical engineering from Sharif University of Technology, Tehran, and his PhD in biomedical engineering from ANU.

He received an Australian Postdoctoral (APD) Fellowship in 2009 and was the recipient of a Fulbright scholarship in 2008. He has more than ten years of industry experience in the ICT sector and worked as the CTO of GPayments Pty. Ltd between 2001 to 2007. His research interests include medical image analysis, high performance computing, and wireless communications.

Prof. Richard Hartley is the Head of Computer Vision and Robotics group at ANU. Prof. Hartley is a Fellow of Australian Academy of Science and an IEEE Fellow. From 1984 until 2001 he was a member of the research staff at General Electric R&D Center in New York where his research involved Computer Vision and Medical Imaging.

He has 34 patents including 9 in the area of medical imaging. He has published more than 200 papers. He is a highly cited author and his book 'Multiple View Geometry in Computer Vision' is the main reference in this area. Prof. Hartley is on the editorial board of IJCV and is the General chair of ICCV 2011. He is the recipient of GE's Dushman award in 1990 (highest recognition for research at GE).

2.4 Contributions

The series of papers [1],[2] and [3] use a simple ray-based model for ultrasound simulation and achieve real-time performance. We want not only to accelerate the simlation by using the GPU but also to use a wave-based model of ultrasound based on the concepts of linear acoustics and spatial impulse response.

More precisely, the implementation has been done in C++/CUDA (Compute Unified Device Architecture). CUDA gives developers access to the native instruction set and memory of the parallel computational elements in CUDA GPUs. We present here an implementation of the computation of the spatial impulse response for polygonal aperture transducers.

We have used an incremental algorithm more suited to the GPU than the one proposed in [4]. The final contribution is the application UltraCuda, developed on C++/CUDA which computes the spatial impulse response of transducer made of many simple aperture.

2.5 Summary and key words

Key words : Simulation, Real-Time, Ultrasound, Spatial Impulse Response, GPU.

The goal of the internship was to develop a linear wave-based simulation of ultrasonic fields. The theory was based on the Tupholme-Stepanishen formalism explained in the Jensen course for calculating pulsed ultrasound field. The Field II Simulation Program developed at the Technical University of Denmark does that simulation but the program runs slowly due to the fact that it runs only on the CPU. It is getting older as it was released in 1994. Designing an algorithm and implementing an application which parallelizes the computation of the spatial impulse response on the GPU were the main goal of my internship. More precisely, my tasks were to

- Become familiarized with ultrasound physics and GPU programming.
- Have a first go with the simulation through Matlab.
- Devise an algorithm suited to GPU programming.

- Develop an application able to simulate different kinds of configurations and the corresponding data structures. The application is called through Matlab commands but all the simulation are made with C++/CUDA programming.
- Verify the validity and time the execution of our implementation.

3 Background

3.1 Overview of Wave-based Ultrasound Modeling

Introduction

Wave-based ultrasound modeling is necessary to improve and to optimize ultrasound imaging. Indeed, by simulating the behaviour of ultrasound devices, it is then possible to design ultrasound imaging devices at a lower cost. The overview of what is wave-based ultrasound will fall into three parts. First, we give an introduction about the different aspects of ultrasound imaging. Then we present what a transducer is. To finish, we present the model we have used for the simulation.

3.1.1 Ultrasound Medical Imaging

Ultrasound imaging remains a very convenient modality. Indeed, it is real-time with high temporal resolution and harmless. Furthermore, ultrasound devices are relatively cheap and portable. The main drawback is the image quality as you can see in the figure 2.

Ultrasound imaging principle is quite easy to understand. For diagnostic ultrasound the recorded image is based on reflected energy. The single device that generates the ultrasound wave and subsequently detects the reflected energy is called the transducer. Figure 1 illustrates how ultrasound imaging works.

An ultrasound wave is directed into the body to interact with tissues in accordance with the characteristics of the targeted issues. The type of interactions that occur are simular to wave behaviour observed with light: reflection, scattering, diffraction, divergence, interference and absorption. That is why previous work has been made to simulate ultrasound on the GPU with a ray-based model, [1], [2], [3]. Meanwhile, the goal of the project is developping a linear wave-based model in order to have a model with a strong reliability. We will introduce now what will be the method used for the simulation and from where it comes from.

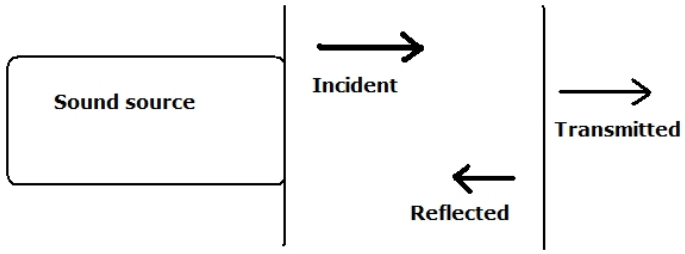

Figure 1: Reflection caused by a sound wave striking a large smooth interface at normal incidence

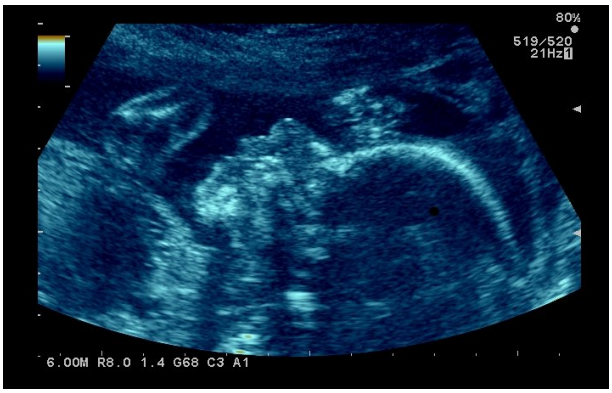

Figure 2: Application of ultrasound imaging, echography of a baby

3.1.2 Transducers

As explained before, a transducer is a device able to convert electricity energy into ultrasound energy and vice versa. This phenomenon is explained by the piezoelectric effect. A single element transducer is a planar geometric surface which vibrates. We will suppose that the vibration is homogeneous on the whole surface, even if it is apodized in reality.

A transducer can be made of multiple simple transducers arranged into a linear array for example. With that configuration, you can set a delay for each aperture and so on having the possibility to modify the shape of the ultrasonic field created.

3.2 The Spatial Impulse Response

Our simulation of the ultrasound wave is based on the same model as the one used in the **Field** program. It relies on the concept of spatial impulse response developed by Tuphole and Stephanishen [4]. It is inherited from the electrical linear system theory. Indeed, a linear electrical system is fully characterized by its impulse response. The output of any $y(t)$ to any kind of input signal $x(t)$ is given by

$$y(t) = h(t) \star x(t) = \int_{-\infty}^{+\infty} h(\theta)x(t-\theta)\,d\theta \tag{1}$$

where $h(t)$ is the impulse response and $\star$ the convolution operator. The transfer function of the system is the Fourier transform of the impulse response.

Fig 3 illustrates the basic set-up of a linear acoustic system. The transducer is rigid and set at position $\vec{r_2}$. It radiates at the sound speed c into a homogeneous medium of density ρ_0. Huygen's principle assimilates each point of a radiating surface as the origin of an outgoing spherical wave. An excitation of the transducer with a Dirac function (δ) will give rise to a pressure field. We

define the accoustic response as the measured response at point $\vec{r_2}$. The response is dependant to $\vec{r_2} - \vec{r_1}$. This is the reason why we characterize it as spatial. Each outgoing waves is given by:

$$p_s(\vec{r_1}, t) = \delta(t - \frac{|\vec{r_2} - \vec{r_1}|}{c}) = \delta(t - \frac{|r|}{c}) \tag{2}$$

Jensen provides then in [4]

$$p(\vec{r_1}, t) = \rho_0 \frac{\partial v_n(t)}{\partial t} \star h(\vec{r_1}, t), h(\vec{r_1}, t) = \int_S \frac{\delta(t - \frac{|\vec{r_1} - \vec{r_2}|}{c})}{2\pi|\vec{r_1} - \vec{r_2}|} \, d\theta \tag{3}$$

where $h(\vec{r_1}, t)$ is the spatial impulse response at the point defined by $\vec{r_1}$. It is then derived as

$$h(\vec{r_1}, t) = \frac{c}{2\pi} \sum_{i=1}^{N(t)} \Theta_2^{(i)}(t) - \Theta_1^{(i)}(t) \tag{4}$$

where $N(t)$ is the number of arc segments that cross the boundary of the aperture for a given time t and $\Theta_2^{(i)}\Theta_1^{(i)}$ the bounding angles as shown in Fig 9. The algorithm to compute the SIR can be summed up in the following steps

- Finding the arc inside the aperture
- Summing the arc lenghts
- Scaling the sum by $\frac{c}{2\pi}$

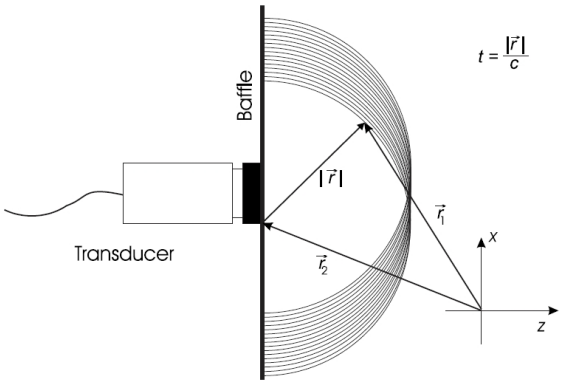

Figure 3: Illustration of the set-up and Huygen's principle. Image source [4]

A detailed description of the algorithm used is given in section 4.4.1. The spatial impulse response is the key of simulating ultrasonic field. The physical problem is finally derived into a geometric problem. The simulation is made on a large amount of points and we can guess that such a geometric algorithm comes up with a lot of arithmetic operations.

GPU programming is a very good way to speed-up the processing of arithmetic operations. Parallelizing the problem is clever due to the fact that we will repeat the same algorithm for a lot of points. That is the reason why the CUDA technology is suited to this simulation. We continue with a brief CUDA introduction.

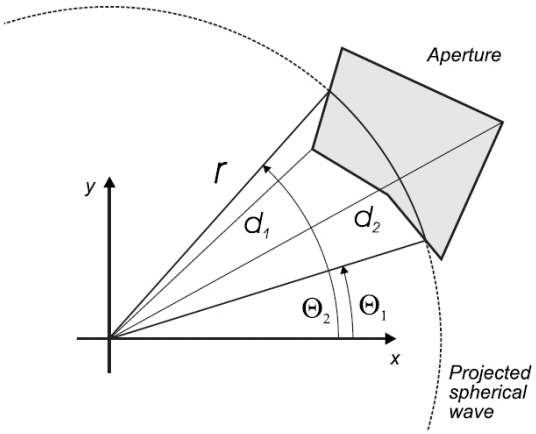

Figure 4: Intersections of the projected spherical wave and the aperture. Image source [4]

3.3 Overview of GPU Programming on CUDA

Introduction

For more than two decades, standard microprocessors were based on a single central processing unit(CPU), such as the Intel Pentiun and AMD Opteron. Recently, the processor manufacturers have changed their strategy due to the lack of improvement in clock frequency and switched to multiple processing units. Indeed, parallelization is an efficient way to increase the computational capacity.

The graphic processing unit(GPU) such as the NVIDIA or ATI families were mainly used for graphic purposes. For example, GLSL is a C-style language for programming shaders which replaces some steps of the graphic pipeline. The power of GPU has never stopped increasing. Nowadays, GPUs are useful not only for graphic purposes but also general purpose arithmetic computing.

GPU has evolved into a highly parallel, multithread, manycore processor with high memory bandwith. That's why it is well-suited to address problems that can be solved with data-parallel computations. The main idea of GPU programming is that data-parallel processing maps data elements to parallel processing threads. The same program will be executed for each data element.

One can compare the characteristics of the CPU and the GPU. For instance, an Intel Core 2 (2.66GHz) provides 14.2 GFlops although a GeForce 9800 GTX (only 675 Mhz) provides 420 GFlops. But the GPU is really efficient if used on a suitable problem. In November 2006, NVIDIA introduced CUDA which is a general purpose parallel architecture which solves many computational problems in a better way than the the CPU does.

3.3.1 Architecture of a CUDA Capable Processor

To begin, only NVIDIA processor are CUDA capable. Using CUDA without a NVIDIA device is still possible through an emulator. A brief description of the architecture of a modern NVIDIA GPU is given here. The two most importants aspects of the architecture are the processing units and the different types of memory.
Figure 5 shows the architecture of CUDA-capable GPU. It is organized into an array of streaming multiprocessors (SMs). Each SM is made of streaming processors (SPs) which share control logic and instructon cache. The number of SPs per SM varies for each device.

Each GPU comes with **global**, **constant** and **texture memory**. Global is the biggest chunk of memory on the device. This memory is accessible from the device and the host. Its main drawback is its low bandwidth. Constant memory is a very fast memory but very size-limited. Each SP comes with its own **shared memory**. Shared memory is only accessible from a SM and benefits of a high bandwith.

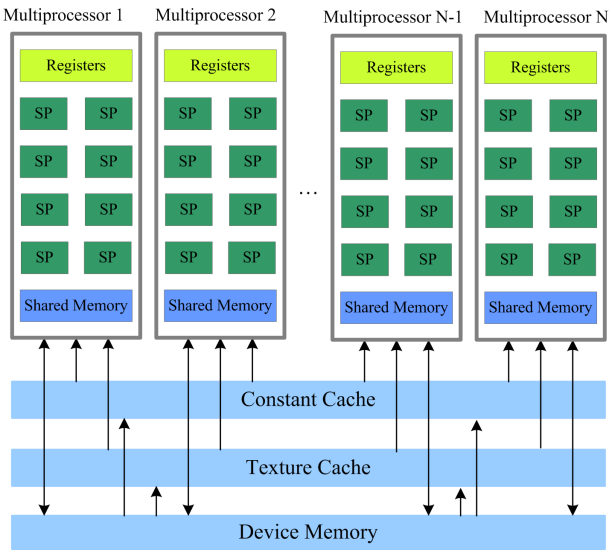

Figure 5: Architecture of a CUDA GPU

3.3.2 Programming model

To begin, it is necessary to define some vocabulary specific to CUDA programming.

- The **host** is the CPU. It gives the order to the **device** (GPU) to make computations.
- A **kernel** is a C-function executed on the device N-times in parallel by N instances.

- A **thread** is one of those instances.
- A **grid** is made of blocks. Each blocks is made of threads. The dimension of a grid may be up to 2 and the dimension of a block may be up to 3.

To improve the understanding of CUDA programming, a link can be made between the software and the hardware. The hardware parts are associated to their abstract part in CUDA.

Hardware	Software
Streaming Processor	Thread
Streaming Multiprocessor	Block
the whole device	the grid

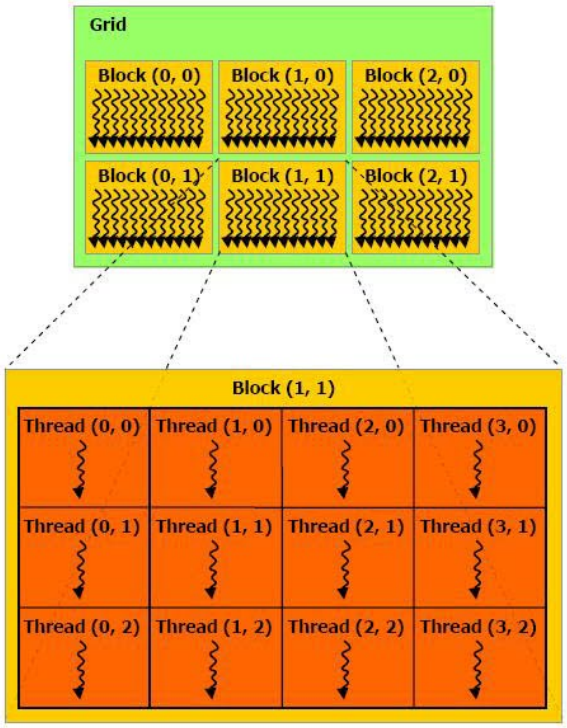

Figure 6: CUDA organization

The organisation in threads-blocks-grid is showed in figure 6. CUDA models the GPU as a coprocessor helping the CPU. When an application executed on the CPU reaches a point where a large amount work needs to be done (for example, a loop computing a large matrix-matrix multiplication), the work is sent to the GPU to accelerate it. Thus, the program generally follows the following steps.

1. The data needed is copied on the memory GPU.

2. The programmer configures the size of the grid and the size of the blocks.
3. The parallel computing is executed through a kernel on the GPU.
4. Once the work finished, the result is copied back to the CPU and application continues.

3.3.3 Goals of a CUDA programmer

A CUDA program is not a trivial task. [8] gives the fundamental things to in CUDA. The priorities of a CUDA programer would be:

- Maximizing parallel execution
- Optimizing memory usage to achieve maximum memory bandwidth
- Optimizing instruction usage to achieve maximum instruction throughput

3.4 Objectives after analysing the problem

The goal of the project will be developping an application able to compute the spatial impulse response for multiple apertures transducer on a large set of points. The number of points n is expected to be more than one million. The expected speeding-up is expected as ten times as what the CPU does. The aperture we will focus on is a linear array transducer. The emitted wave is produced by an array of rectangular sound sources 7.

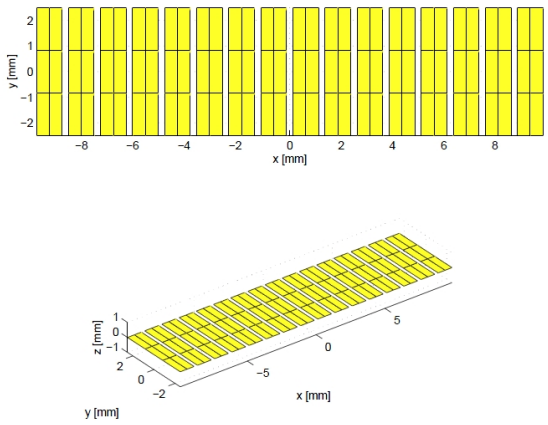

Figure 7: Linear array transducer from the user guide of field II, [5]

4 Development of the application

4.1 About UltraCuda

UltraCuda is the name of the application we have implemented to solve the problem exposed before. The application is made for a Windows environment and Matlab is necessary to use it. First, we explain precisely how the problem can be parallelized. Then we present our algorithm which computes the spatial impulse response. To finish, we will deschribe the final application architecture. We briefly introduce now the working environment and then the steps of the development.

We have first used Matlab for testing the geometric algorithm. Indeed, it is quicker to set up a basic simulation on Matlab than in C++. It is then easy to debug and it is useful to test the algorithm. The debugging in C++ will be then about memory management on the host and the device.

Then, we have used Matlab to call the C++/CUDA function through a wrapper created with the mex library. It is a convenient way to manipulate vector outside C and manipulating data before sending them to our program. CUDA has been chosen for the multi threading. Even if OpenCL can be used with other GPUs than the NVIDIA ones, CUDA remained very convenient. We had already a lot of helper and generic algorithms like reduction already implemented in CUDA and fully tested.

4.2 Chronology of the developement

We will sum up here the step of the development. The steps of the develoment were:

- Simulating the Spatial Impulse Response for a simple transducer on Matlab and visualizing the results by different kinds of plottings.
- Discussing the architecture of the application and the data structure used
- Implementing the inititialisation of the algorithm.
- Designing algorithms suited to GPU computation.
- Implementing the new algorithm for a simple transducer.
- Implementing two versions of the simulation for multiple transducers.

4.3 Architecture of the application

The program consists of a C++/CUDA program. All calculations are performed by the C++ program and all the data is kept by the C++ program. All the massive parallel part of the code are executed by CUDA. The C-function are called through a wrapper. The organisation of the program is shown on the figure 8.

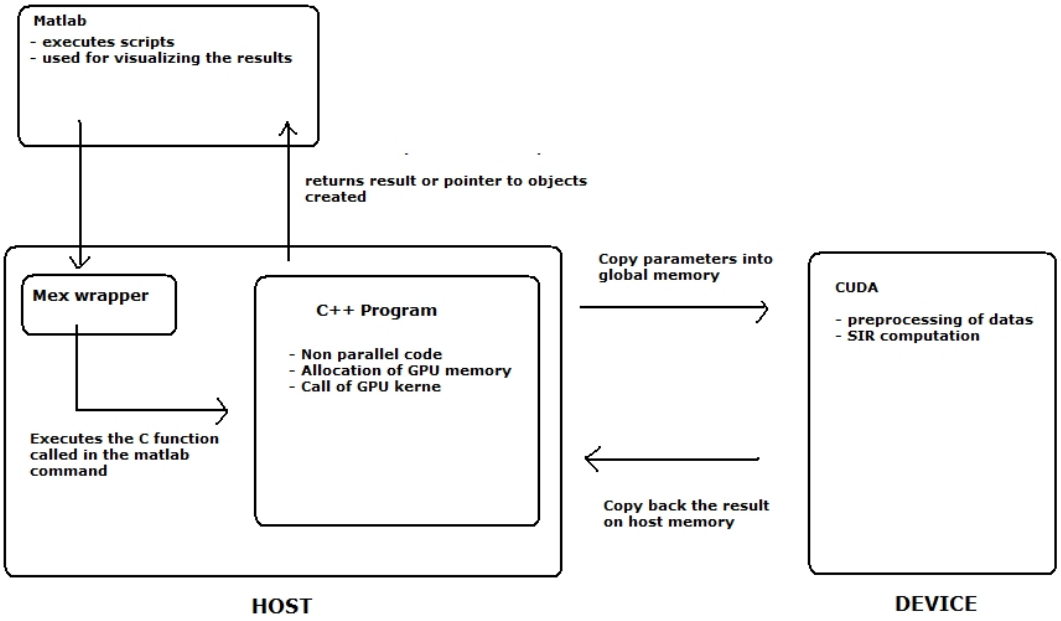

Figure 8: Architecture of the application

4.3.1 Multi-threading

The computation is performed on a large number of points. The way of parallelizing the problem comes up quite naturally. It is based on points. Each point is assigned to a thread. All the field points and the aperture points are copied on the global memory of the GPU. Then, kernels are executed and the results are copied back to the host memory.

To efficiently use the capacity of the GPU, to devise a new incremental algorithm for computation of the spatial impulse response. First we will show how to compute the Spatial Impulse Response with an incremental algorithm and then the two ways of prepocessing the data.

4.4 Computing the Spatial Impulse Response for one transducer element

4.4.1 Efficient algorithm for computing the intersection between simple polygons and a circle

The physical problem we have introduced before is actually a geometric problem. It will use some classic problem of computational geometry such as

- Localisation of a point
- Intersection
- Convexity problem

Our goal is to sum the angular lengths of the arcs lying inside the polygon. This point is essential to compute the spatial impulse response of the ultrasound wave. Jensen proposes to find all the intersections, sort and sum them minus some verifications.

That method is not suitable for the GPU. In order to sort an array, you need to dynamically allocate an array on the shared memory. The maximum of threads per block is 512. If you need to sort 100 intersections per thread you will need $10 * 512 * 32 = 163840$ bytes per block. This far exceeds the amount of shared memory available per block (e.g. 16384 bytes for GT200 architecture). This is the reason why we have developed an incremental algorithm to adapt the theory to GPU computation.

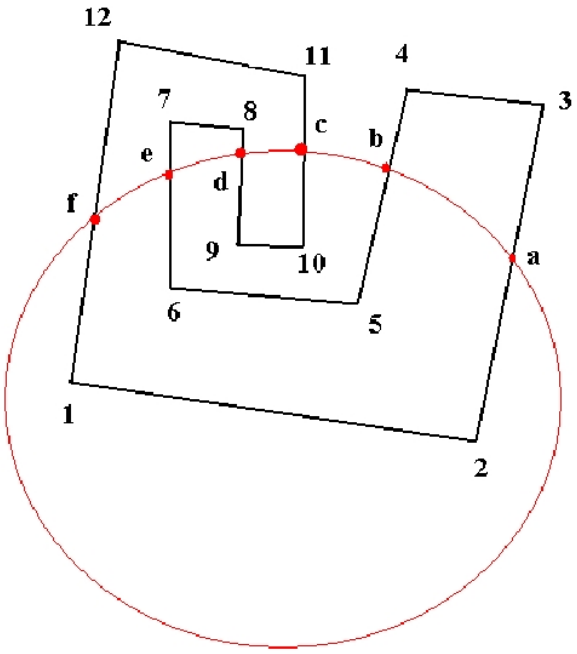

Figure 9: A simple polygon intersection

4.4.2 Initializing the algorithm

Starting from point 1 of the polygon, go to the first intersection you find. Let θ_0 be the corresponding angle. This point will be set as the angular origin. We set the discontinuity on that point by applying the following transformation to the next found angles.
Let θ be the angle of the current intersection.
if $\theta > \theta_0$, then

$$\theta' = \theta - (\theta_0 + \pi) \tag{5}$$

else if $\theta < \theta_0$, then

$$\theta' = \pi - \theta_0 + \theta \tag{6}$$

4.4.3 Exploring all the edges

We explore the polygon in the counterclockwise order.
Let S be the sum of the arc lengths.
Let P be the array of points of the polygon given in the counterclockwise order.
Let n be the number of points in the polygon .
Let $sign$ be an integer whose value alternate between 1 and -1.
Let θ_{min} be the angle of the closest intersection to the first one intersection in the counterclockwise order.

- To initialize the algorithm, Find the first intersection and keep the first angle θ_0. For the other intersection angles, apply the transformation presented in the previous section.
  $sign = -1$;
  $S = sign * \theta_0$;
  $sign = -sign$;

- Explore all the edge in counterclockwise order. For each edge, find the intersections. If there are 2 intersections, proceed first with the closest intersection to the first point of the edge. Let θ be the corresponding angle after the transformation.
  $S+ = sign * \theta$;
  $sign = -sign$;
  if ($\theta - \theta_0 < \theta_{min} - \theta_0$) then $\theta_{min} = \theta$;

- Test if the arc made of the first intersection and the closest to it lies inside the polygon. If not $S = 2 * \pi - S$;

4.4.4 Special cases, tangents and corners

There are some special cases in the algorithm. One may have to deal with a tangent intersection or a vertice intersection. The solution is to ignore the vertice intersections and the tangent intersection. One can notice that a circle contained in a square with tangent intersections at each edge will be problematic. We avoid that case by the following method. We precompute before the minimal distance from the circle center to the polygon. If the center is inside the polygon and the radius of the circle is less or equal to d_{min}, it means that the whole circle is inside the polygon. Thus the returned value will be 2π.

4.4.5 Geometric elements

The implementation of the algorithm relies on smaller algorithms of computational geometry. For example, we have implemented those algorithms:

- test wether a point lies on the left side of an directed line.

- test wether a point lies inside a polygon.

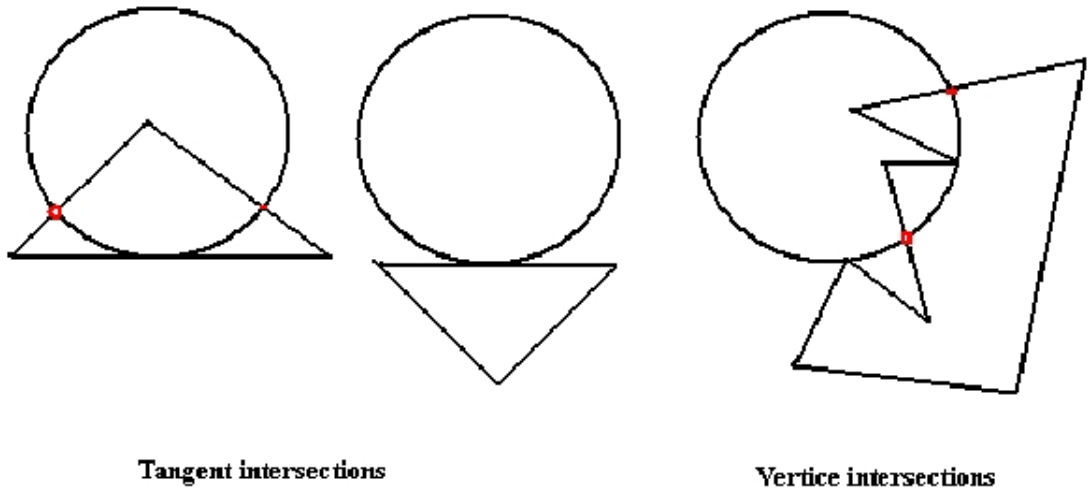

Figure 10: Special cases

- computing the minimal distance from point to a line segment.

Especially, one of the crucial point is testing the position of an arc.

Let θ_1 and θ_2 be the angle of 2 consecutive intersections such that $\theta_1 < \theta_2$. We claim that the arc is inside if any points of angle θ such that $\theta 1 < \theta < \theta_2$ is inside the polygon. Testing only one point between the 2 intersections is sufficient, the most convenient would be the midpoint whose the angle is $\theta_{mid} = \frac{(\theta_1+\theta_2)}{2}$. This test is suspected to be a source of bugs due to a lack of precision in the computation when the distance between points is really small.

4.5 Preprocessing of the data

The previous algorithm is very useful. Meanwhile, when you use many apertures, it may be beneficial to preprocess the data. Preprocessing the data is useful to speed-up the computation. Indeed, we compute the time boundaries of the SIR for each field points. To compute the time limits, we only need to compute the minimal and maximal distance from a point to an aperture. Then we convert that distance into the correct time sample. By doing that, we avoid to execute a code which will return 0 a lot of times. There is two way of doing that. Note that a transducer is made of many single transducer elements.

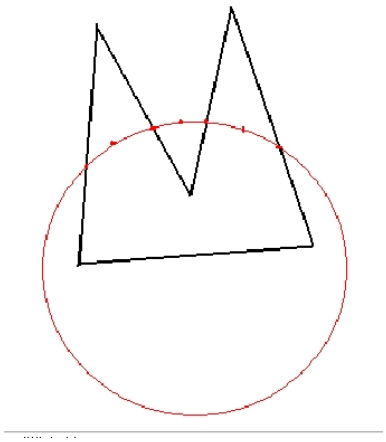

Figure 11: Midpoints of different arcs

4.5.1 First algorithm

We compute the time limits for the whole transducer. We keep the start time and the end time for each field points for the whole transducer in memory . This algorithm does not use a lot of memory but it is clearly not the fastest one.

4.5.2 Second algorithm

In that version, we does the same as the previous algortihm but for each single transducer element. This version is quite faster than the previous algorithm. However, the use of memory is $(nAp * nPoint)$ where nAp and $nPoint$ are the number of single transducer element and the number of field points respectively. It reduces the number of points you can process with a function call. Meanwhile, it is possible to split your point set and loop on the subdivision created. The figure 12 illustrates the two previous algorithms.

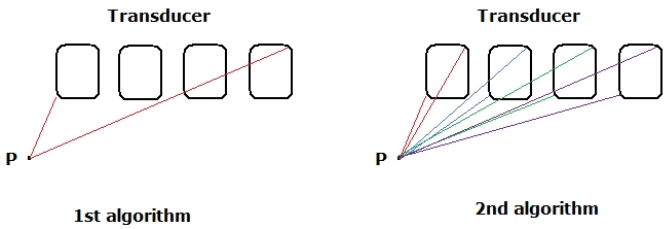

Figure 12: Computing the time boundaries

5 Results

The machine I used was not CUDA capable so I connected to a CUDA capable machine with VNC viewer. As you can not debug GPU code, I debugged and tested the correctness of my CUDA code in emulator mode. It allows you to debug the code as it is emulated and runs like a C program.

5.1 Validation

In this version we present our results and discuss their validity compared to Field II and explain what are the new directions to take in order to achieve entirely the project.

5.1.1 Comparaison with Field II

To test our results, we have first compared our simulation to the figures contained in **??**. We have plotted the spatial impulse response for differents points on a line contained in a parallel plane to the transducer 13 14.

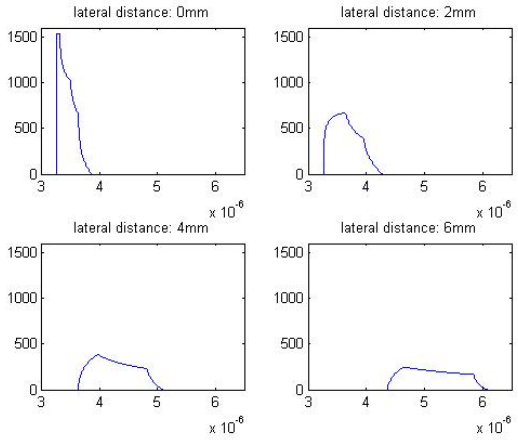

Figure 13: Spatial Impulse Response for 4 points on a line

5.1.2 Testing different kinds of configuration

With the matlab interface, you can simulate the evolution of the spatial impulse response on a grid parallel to your planar transducers. It gives the following results. One can notice that it is wonderful pictures but it is also important to verify some basics things like the symetry of the images produced. By making an animation and vizualizing, we have been able to detect bugs that we did not detect before. The following figures show the kind of configurations we tested,15,16,17,18,19,20.

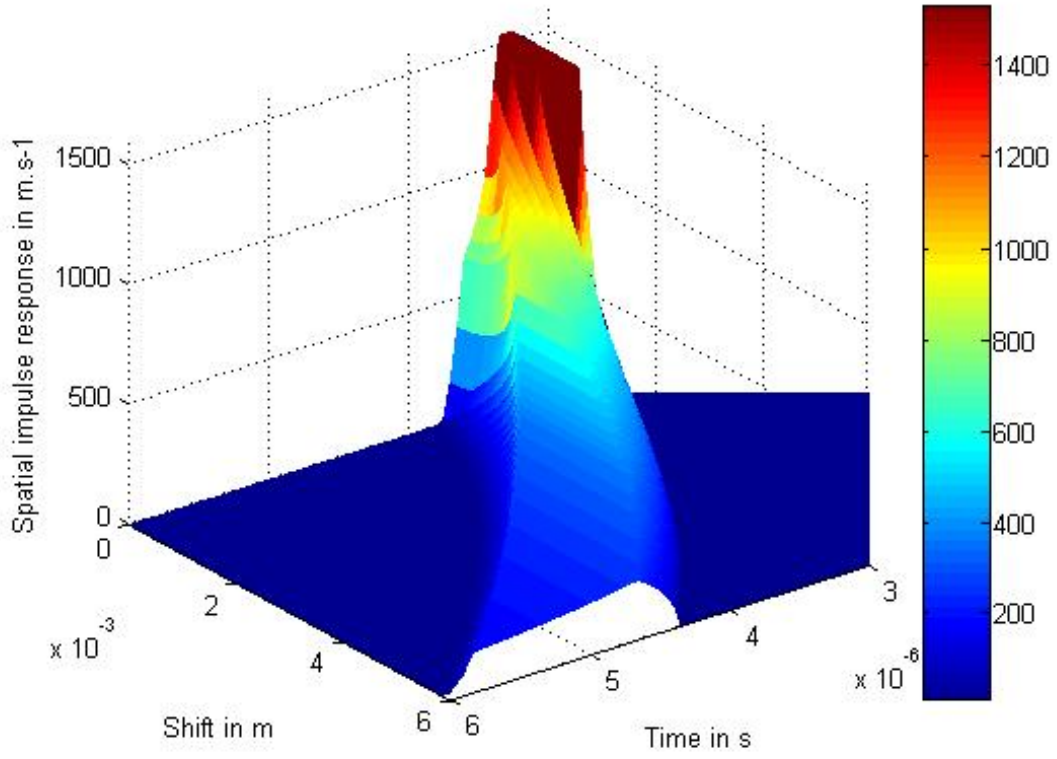

Figure 14: Spatial Impulse Response for points on a line

With 2 apertures, we obtain the following pattern.

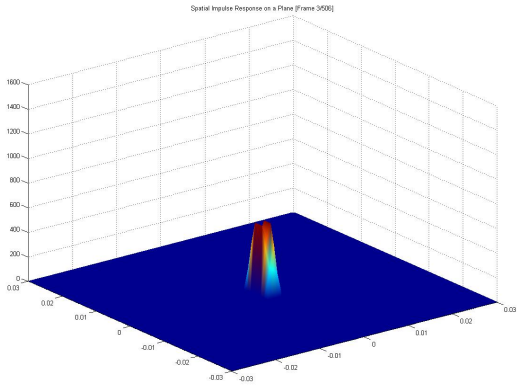

Figure 15: 2 transducers

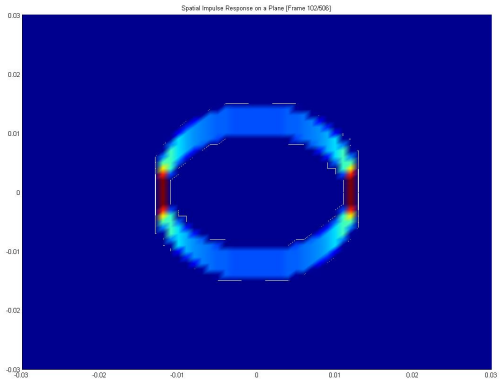

Figure 16: 2 transducers - Top view

Then with 16 apertures, the contributions of each apertures are summed,

To finish, we have tested with 128 apertures, a classic configuration for real transducers.

5.1.3 Comparaison with Field II

Field II documentation specifies that it uses double precision. Only very recent GPUs are actually supporting double precision. Mine was none of them. This is the reason why we decided to compare our results with what Field II gives. We were very surprised to observe a difference

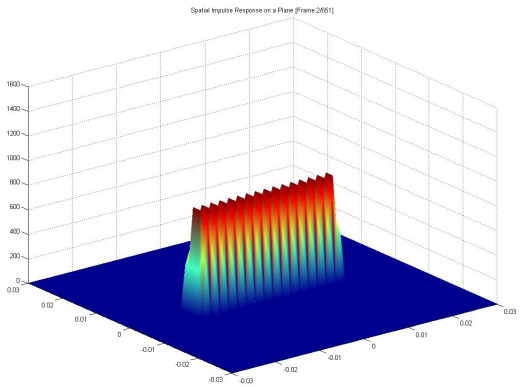

Figure 17: 16 transducers

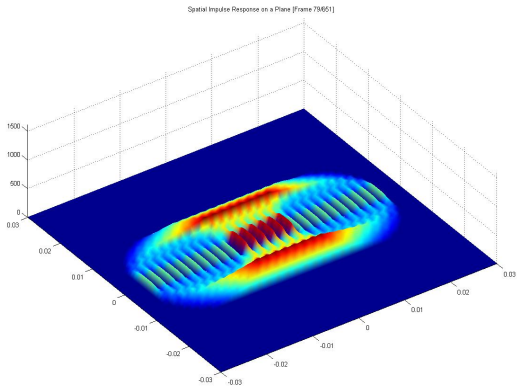

Figure 18: 16 transducers - Top view

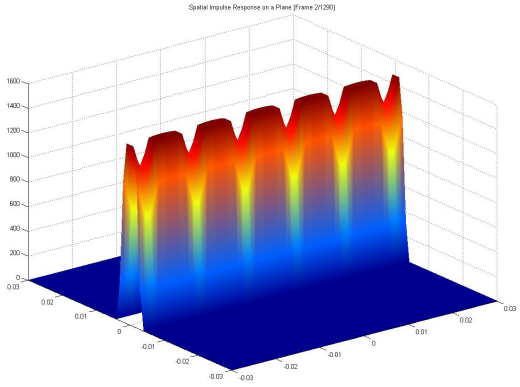

Figure 19: 128 transducers

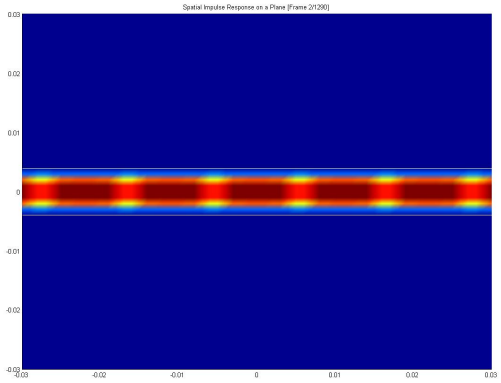

Figure 20: 128 transducers - Top view

between our results when we substract the response given by UltraCuda and Field II.

By doing that, we have realized that Field II uses approximations for giving satisfying results. Ours is a direct implementation of the theory and so on does not use any approximation. It is then still faster than what Field II does. The Matlab scripts we have used for comparing the two programs are given in the appendix.

5.2 Performances of the application

We have benchmarked our implementation on a work station equiped with a NVIDIA GeForce GTX 295, 8 core Intel Core i7 2.93 $2.93\ \mathrm{GHz}$ and 6GB RAM.
We compared what the results obtained with our program and field II. We measured the time needed to compute the response. We first played with the number of apertures (transducer elements) and then with the number of points processed by the application.

The first algorithm is much slower than the second but uses less memory on the GPU, especially when the number of aperture increases. Indeed, the used memory size is $(nPoint)$ whereas it is $(nPoint * nAperture)$, where $nPoint$ is the number of points used for the computation and $nAperture$ is the number of apertures in the transducer.
In some test, only the first algorithm appears. The reason is that the second one goes over the chunk of constant memory available and crashes the program.

5.2.1 Timing by varying the number of aperture

We have used different powers of 2 for the number of apertures.

Even if the 1st algorithm is slower, there is still a good speeding-up during the computation.

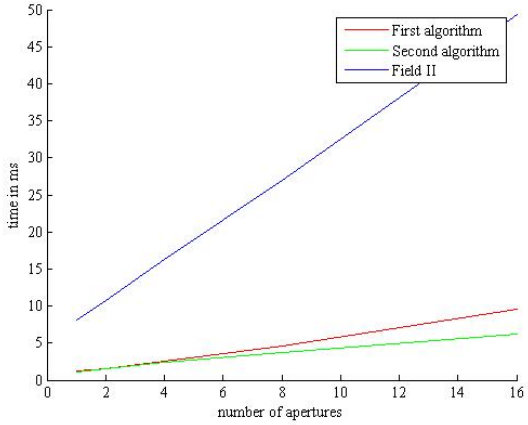

Figure 21: Timings by increasing the number of apertures

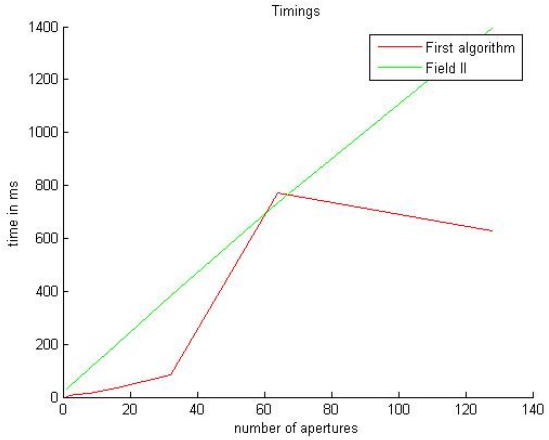

Figure 22: 1st algorithm against field II - 14424101 points

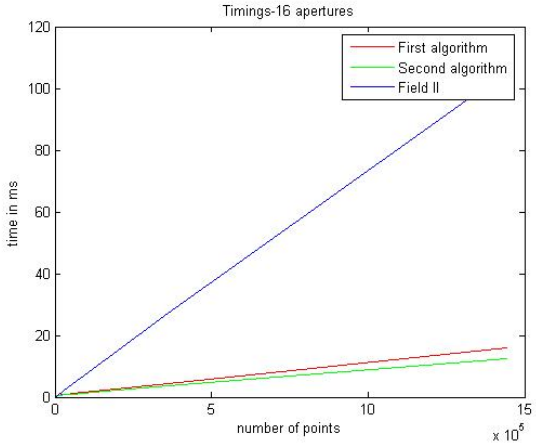

Figure 23: Timings by increasing the number of points - 16 apertures used

5.2.2 Timing by varying the number of points

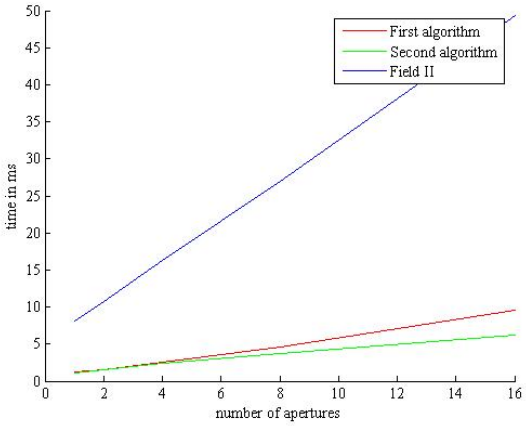

Figure 24: Timings by increasing the number of points - 64 apertures used

The other parameter which can change the time computation is obviously the number of field points. Here are the results we have found when we increase the number of field point. The results are shown in figure 23,24.

5.3 Incoming tasks and possible improvments

The biggest unsolved problem is that the application gives false results when it tests the position of points very closed to each other. The validity of the algorithm is correct but the computation in some case ends to a false result. We did not have the time to fix that error. The reason is that the float precision is too small.
The main issue of that bug is that it leads to some peaks in the display of the spatial impulse response and will probably cause some image artefacts.

I was not really familiar with CUDA before the internship, and I still feel that I have a lot of things to learn about CUDA programming. I have thus used a simple parallelization on points. Meanwhile, a very efficient way to parallelize the processing would be to:

- Assign a point to a block.
- Assign a time sample to a simple thread.
- Loop on all the time samples to compute the spatial impulse response for each point.

Another improvment would be optimizing the code so that it uses less registers. Then, minimizing the use of structures like if, then else which slow the multi-threading by causing divergency.

6 Conclusions

6.1 About UltraCuda

UltraCuda shows that it is possible to benefit of the speeding-up of the GPU by using the CUDA technology. Meanwhile, what we have developed is still a prototype which can be optimized. Computing the Spatial Impulse Response is a key step for simulating ultrasonic field and UltraCuda does it.

Many things remains to be done. Once the basic prototype is fully optimized, it will be necessary to include all the kind of apertures possible and probably coming up with a new interface would be a great improvment to our application compared to Field II. Indeed, it can only be used through a Matlab interface like UltraCuda.

6.2 Personal Experience

For the project, I needed many skills that I learned during my internship and I realize now that it was useful. I feel very happy to have been able to find an application to the courses I attended during my two years at the ENSIMAG and one semester at the UPC Barcelona. Especially, Computational Geometry, Algorithmics, Computer Architecture, MultiThreading were the notions I have applied for the development of the project.

After all, I have discovered that Matlab was not only a source of pain but also a very convenient way of manipulating arrays and quickly visualizing data without implementing an interface through QT for exemple. I have always been curious about GPU programming and I feel very happy with having my first go with CUDA on such an interesting project.

Apart from the technical and theorical learning, I reckon that this internship made me a better English speaker and probably with a language tainted by some Australian idiomatics. Meeting so many people from different backgrounds and especially Asia has enlarged my view of the world. Discovering Australia was really a worthful experience and it would be very hard to forget all the amazing places I have visited there.

Unfortunatly, all good things come to an end and I will return to Europe with new perspectives which are continuing in doing research in Image processing and Vizualisation and I hope that my upcoming career will be strongly related with the growing area of GPU programming.

References

[1] Oliver Kutter, Ramtin Shams, Nassir Navab *Visualization and GPU-accelerated simulation of medical ultrasound from CT images*. Computer methods and programs in biomedicine 19 December 2008

[2] Ramtin Shams, Richard Hartley, Nassir Navab *Real-time simulation of medical ultrasound from CT images*. RSISE, NICTA, CAMP(Computer Aided Medical Procedure)

[3] Tobias Reichl, Josh Passenger, Oscar Acosta, Olivier Salvado *Ultraound goes GPU: real-time simulation using CUDA*.

[4] Jorgen Arendt Jensen *Linear description of ultrasound imaging systems*. Notes for the International Summer School on Advanced ultrasound Imaging Technical university of Denmark Release 1.01, June 29, 2001

[5] Jorgen Arendt Jensen *User' guide for the Field II program*. Notes for the International Summer School on Advanced ultrasound Imaging Technical university of Denmark August 17, 2001

[6] David B. Kirk, Wen-mei W. Hwu *Programming massively parallel processors*. Morgan Kaufman, 2010

[7] NVIDIA *CUDA Programming guide version 2.3*. 2010

[8] NVIDIA *CUDA Best practices guide version 3.0*. 2010

[9] Thibault Cuvelier *Une introduction à CUDA*. developpez.com

[10] Hedrick Hykes Starchman *Ultrasound physics and instrumentation*. Evolve

[11] Philip E. Bloomfield *Extensions of the scattering-object function and the pulser-receiver impulse response in the field II formalism*. Elsevier 21 July 2004

[12] Marshall Cline *C++ FAQ LITE*. A very good website about frequently asked questions in C++.

[13] *Dr Ramtin Shams website*. `http://users.cecs.anu.edu.au/~ramtin/`

[14] *Pr Richard Hartley website*. `http://users.cecs.anu.edu.au/~hartley/`

7 Appendix

7.1 Résumé du stage

L'objectif du projet était de développer une simulation des champs ultrasoniques à partir d'un modèle linéaire d'onde. La théorie est basée sur le formalisme de Tupholme-Stepanishen. Le programme de simulation Field II développé à l'Université Polytechnique du Danemark effectue cette simulation mais l'exécution du programme est longue car il utilise seulement le CPU de manière linéaire. Les principales tâches à effectuer étaient donc de créer et d'implémenter des algorithmes adaptés au calculs effectués sur le GPU. Plus précisement, mon stage a consisté en:

- Se familiariser avec la physique des ultrasons et la programmation CUDA.
- Faire une première simulation via Matlab.
- Construire des algorithmes adaptés et optimisés pour la programmation GPU.
- Développer une application capable de simuler différents types de configuration et les structures de donnée correspondantes. L'application est appelée grâce à des commandes Matlab mais toutes les simulations sont éxécutées par le code C++/CUDA.
- Valider le programme et mesurer les temps d'éxécution.

7.2 Geforce GTX 295

The graphic card which I used fot my tests was a Geforce GTX 295. I detail here the capacity of that card.

totalGlobalMem:	939196416
freeGlobalMem:	877543424
sharedMemPerBlock:	16384
regsPerBlock:	16384
warpSize:	32
memPitch:	262144
maxThreadsPerBlock:	512
maxThreadsDim:	[512 512 64]
maxGridSize:	[65535 65535 1]
totalConstMem:	65536
major:	1
minor:	3
clockRate:	1242000
textureAlignment:	256
deviceOverlap:	1
multiProcessorCount:	30
kernelExecTimeoutEnabled:	1

Typically, the gloabal memory is where I stored the preprocessed datas and the results. It limits so the number of processed points.
The registers per blocks is the number of registers per shared processors. The more register you have the best it is because the threads will not be queued. But not using all the registers will slower the execution also.
The maximal dimensions of block and grid gives you what you can expect for a kernel configuration.

7.3 Some Matlab scripts

The following script shows how to create a linear transducer.

```
1
2  %Create a linear transducer centered at the origin of the plane z=0
3  %The direction of the linar transducer is the x Axis
4  %height width are the...height and width of the simple aperture Shape
5  %nElem is the number of elements in the transducer- must be even
6  %Kerf is the space between each simple aperture
7  %field points is your set of field points
8
9  function ptr = linearTransducer(width, height, nElem, Kerf, fdPoints)
10
11 %Computing the left start of the transducer
12 nLeft = floor(nElem/2);
13 nLeft
14
15 %start of the first aperture
16 if (nLeft>0)
17     Xstart = -nLeft*(Kerf+width) + Kerf/2
18 else
19     Xstart = -width/2;
20 end
21 y = height/2;
22
23 % Aperture points. Each column represtion a point in 3-space
24 ap = [Xstart Xstart+width Xstart+width Xstart ;-y -y y y;0 0 0 0 ]
25 % Duplicate the first point of apreture at the end for computational
26 % convinience
27 ap(:,size(ap,2)+1) = ap(:,1);
28
29 %Then we create Cuda object
30 ptr = UltraCuda('createsirt',single(fdPoints),single(ap),int32(nElem),single
       ([1 0 0]),single(Kerf+width));
31 end
```

The following script generates an animation of the propagation of the wave on a parallel plane to the planar transducer.

```
%generate animation for the rectangular aperture
global c
pause on

% Initialise data
% Resolution of the grid in meters
res = 0.0005;
% Define the exents of the field plane
x_ext=[-0.03 0.03];
y_ext=[-0.03 0.03];
[X Y] = meshgrid(x_ext(1):res:x_ext(2),y_ext(1):res:y_ext(2));

% Plane depth where the SIR is computed
z = 0.006;
Z = z * ones(size(X));

%making the grid
fdPoints = [X(:)';Y(:)';Z(:)'];


% Aperture points. Each column represtion a point in 3-space
% ap = (-0.0015 0.0015 0.0015 -0.0015 ;-0.0025 -0.0025 0.0025 0.0025 ;0 0 0 0 );
% Duplicate the first point of apreture at the end for computational
% convinience
% ap(:,size(ap,2)+1) = ap(:,1);

% Initializing the Cuda Object
% ptr = UltraCuda('creategridt',single(X),single(Y),single(z),single(ap),int32(10),single((1 0
    0)),single(0.0005));

%New version with centered aperture
ptr = linearTransducer(1/1000, 5/1000, 10, 1/1000, fdPoints);
UltraCuda('initt',ptr);

% Time boundaries
t = UltraCuda('gettbt',ptr)
T = t(1):t(2);

nFrames = size(T,2) % a ver

v=view(3);
for p=1:nFrames
% Z=UltraCuda('csirgridt',single(T(p)),int32(size(X)),ptr);
    Z=UltraCuda('csirt',single(T(p)),ptr);
```

```
44      %loading image
45      surf(double(X),double(Y),double(reshape(Z,size(X,1),size(X,2))));
46      axis([x_ext,y_ext 0 1600])
47  % axis off;
48      view(v);
49
50      str=sprintf('Spatial Impulse Response on a Plane [Frame %d/%d]',T(p)-T
            (1)+1,numel(T));
51      title(str);
52      colormap(jet)
53      shading flat; shading interp;              % Turn the grid off
54     pause
55      v=view;
56  end
57
58  UltraCuda('cleansirt',ptr);
59  clear mex
```

I used the same kind of script for all the timing.

```
1   function y = benchmark(res)
2   field_init();
3   nAp = [1 2 4 8 16 32 64];
4   for i=1:size(nAp,2)
5       [m1 m2 m3 nfdpoint] = testTiming(1/1000,5/1000,nAp(i),0.05/1000,res);
6       y1(i)=m1;
7       y2(i)=m2;
8       y3(i)=m3;
9   end
10  y = nfdpoint;
11
12
13  %do the plot
14  hold on
15  plot(nAp,y1,'r',nAp,y2,'g',nAp,y3,'b');
16  xlabel('number of apertures');
17  ylabel('time in ms');
18  legend('First algorithm','Second algorithm','Field II');
19  field_end();
20  clear mex;
```

```
function [m1 m2 m3 nfdPoint] = testTiming(width,height,nElem,Kerf,res)

% Initialise data

% freq = 20000000;
%
% Define the exents of the field plane
% Plane depth where the SIR is computed

z = 0.006;
x_ext=[-0.03 0.03];
y_ext=[-0.03 0.03];
[X Y] = meshgrid(x_ext(1):res:x_ext(2),y_ext(1):res:y_ext(2));
nfdPoint = size(X,1)*size(X,2);
X = reshape(X,1,nfdPoint);
Y = reshape(Y,1,nfdPoint);
Z = z*ones(1,nfdPoint);

fdPoints = ones(3,size(X,2));
fdPoints = [X;Y;Z];

%
tic
ptr = linearTransducer(width, height, nElem, Kerf, fdPoints)

%initialise the time limit for every point
UltraCuda('initt',ptr);

% Time boundaries
t = UltraCuda('gettbt',ptr);

%storing the result from my program
T = t(1):t(2);
nStep = size(T,2);
for p=1:nStep
    UltraCuda('csirt',single(T(p)),ptr);
end

%cleaning my memory
UltraCuda('cleansirt',ptr);
m1 = toc;

%

tic
ptr = linearTransducer(width, height, nElem, Kerf, fdPoints)

%initialise the time limit for every point
```

```
49 UltraCuda('initt2',ptr);
50
51 % Time boundaries
52 t = UltraCuda('gettbt2',ptr);
53
54 %storing the result from my program
55 T = t(1):t(2);
56 nStep = size(T,2);
57 for p=1:nStep
58     UltraCuda('csirt2',single(T(p)),ptr);
59 end
60
61 %cleaning my memory
62 UltraCuda('cleansirt',ptr);
63 m2 = toc;
64
65
66 %
67 %doing the same for the field2 program
68 %comparing with the field2 program
69 tic;
70
71
72 %showing time
73 set_field('show_times',1);
74 set_field('c',1548);          %speed of sound in homogeneous medium
75 set_field('fs',20000000) ;          %set the sampling frequency
76
77 focus = [0 0 0];%No focus
78 %
79 %Converting the point into the field2 format
80 fdPoints = fdPoints';
81 Th = xdc_linear_array (nElem, width, height, Kerf, 3, 5, focus);
82
83 for i=1:nfdPoint
84     %computing the impulse response for point i
85     [h tstart] = calc_h(Th,fdPoints(i,:));
86 end
87
88
89
90
91 m3 = toc;
92
93 %clear mex filesx
94
95
96 end
```